CMPS 312

UI Components and Layouts

Dr. Abdelkarim Erradi

CSE@QU

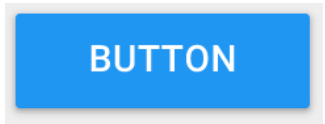
Outline

1. UI Components
2. Layouts

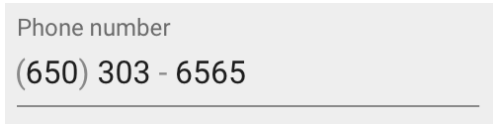
Examples are available @
cmps312-content\examples\05.ui-components-layouts

UI Components

Button

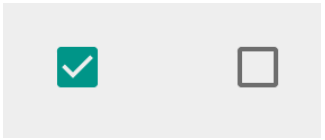


TextField



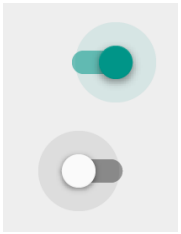
Slider

CheckBox



RadioButton

Switch

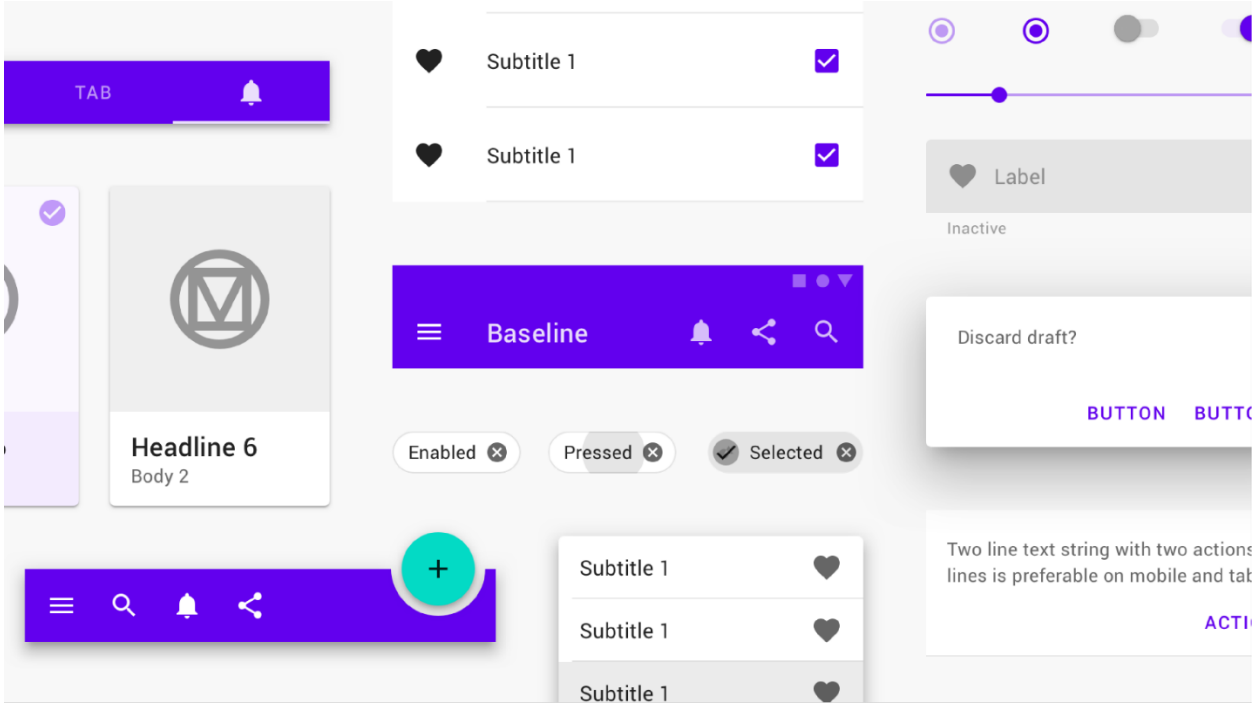


Back

Full list available at this link

UI Components

- Built-in Jetpack Compose UI Components follow Google Material design system https://material.io/
  - Provides a **consistent experience** across all platforms and applications (Android, Web, Flutter, iOS)



Text box

- **`Text()`** displays a simple text

```
Text(
    text = "Jetpack Compose",
    style = MaterialTheme.typography.h4
)
```

Jetpack Compose

```
Text(
    text = "سور القرآن الكريم",
    textAlign = TextAlign.Center,
    modifier = Modifier.fillMaxWidth(),
    style = TextStyle(
        fontWeight = FontWeight.Bold,
        fontSize = 24.sp,
        color = Color.Blue,
        textDirection = TextDirection.Rtl
    )
)
```

سور القرآن الكريم

TextField

- **`TextField()`** collects input from a user. For more styling options, use **`OutlinedTextField()`**

```
@Composable
fun NameEditor(name: String, onNameChange: (String) -> Unit) {
    OutlinedTextField(
        value = name,
        onValueChange = onNameChange,
        label = { Text("Your name") }
    )
}
```

Keyboard options

- TextField lets you set **keyboard configurations** options, such as the keyboard layout, or enable the autocorrect, capitalization, autoCorrect keyboardType
- Also, it allows you to set a visual formatting of the input value, like replacing characters with * for passwords

```
@Composable
fun PasswordTextField() {
    var password by remember { mutableStateOf("") }

    TextField(
        value = password,
        onValueChange = { password = it },
        label = { Text("Password") },
        visualTransformation = PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password)
    )
}
```

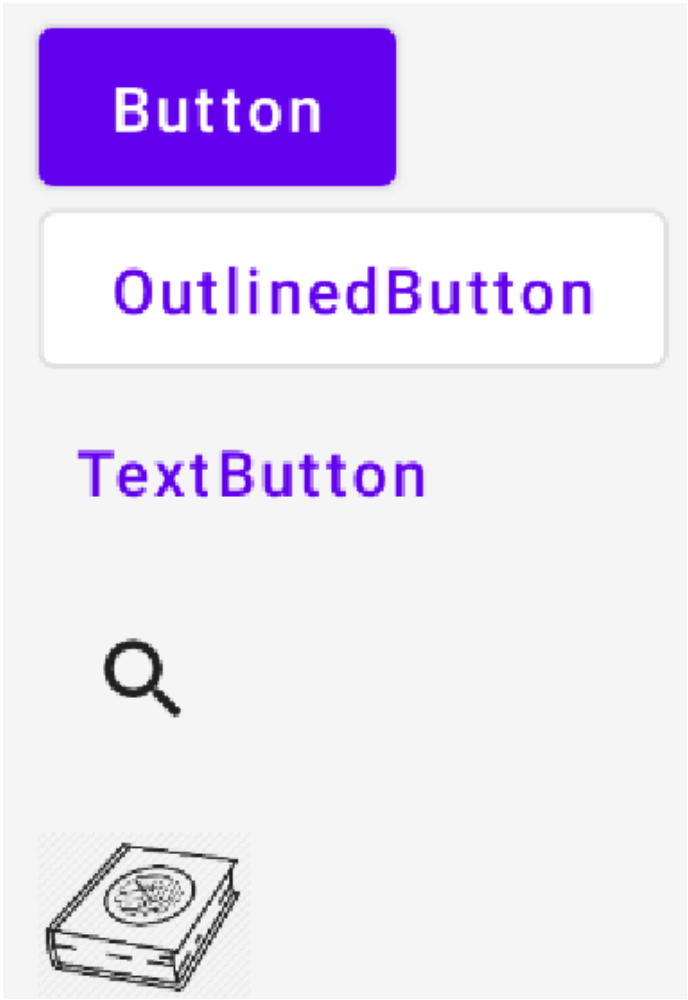
Image

- Displays an image from the res/drawable folder

```
Image(painter =
    painterResource(R.drawable.img_compose_logo),
    contentDescription = "Jetpack compose logo",
    modifier = Modifier.height(300.dp))
```

Button

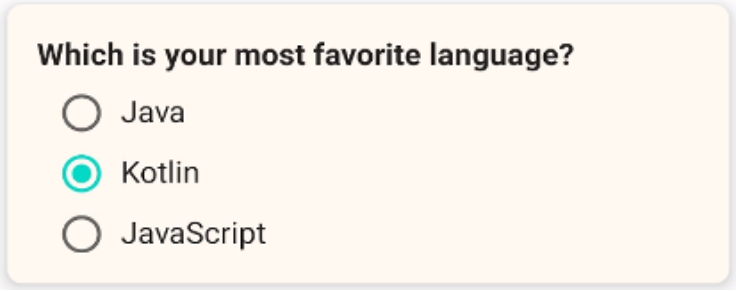
```
Button(onClick = {}) {
    Text("Button")
}

OutlinedButton(onClick = {}) {
    Text("OutlinedButton")
}

TextButton(onClick = {}) {
    Text("TextButton")
}

// Search icons @ https://fonts.google.com/icons
IconButton(onClick = {}) {
    Icon(
        Icons.Outlined.Search,
        contentDescription = "Search",
    )
}

IconButton(onClick = {}) {
    Icon(painterResource(id = R.drawable.ic_quran), "Quran")
}
```



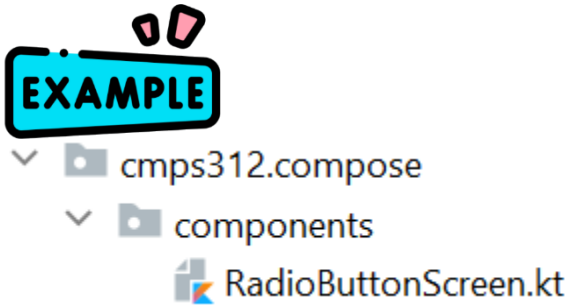
cmps312.compose
components
ButtonScreen.kt

Radio Button

- A Radio Button is used to select a **single** option from a list of options



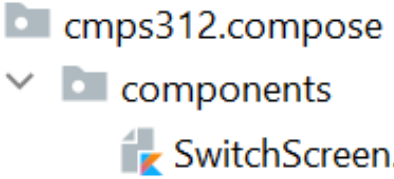

```
radioOptions.forEach { option ->
    Row(
        Modifier
            .fillMaxWidth()
            .selectable(
                selected = (option == selectedOption),
                onClick = { onOptionSelected(option) }
            )
            .padding(horizontal = 16.dp, vertical = 4.dp)
    ) {
        RadioButton(
            selected = (option == selectedOption),
            onClick = { onOptionSelected(option) }
        )

        Text(
            text = option,
            modifier = Modifier.padding(start = 8.dp)
        )
    }
}
```



Switch

- A Switch toggle the state of a single item on or off


```
Row {
    Text(
        text = "Turn on dark theme",
        modifier = Modifier.padding(end = 8.dp)
    )
    Switch(
        checked = isDarkMode,
        onCheckedChange = { isDarkMode = it }
    )
}
```



Checkbox

- Checkbox is used to represent two states i.e., either checked or unchecked

DropDown

Slider

Snackbar

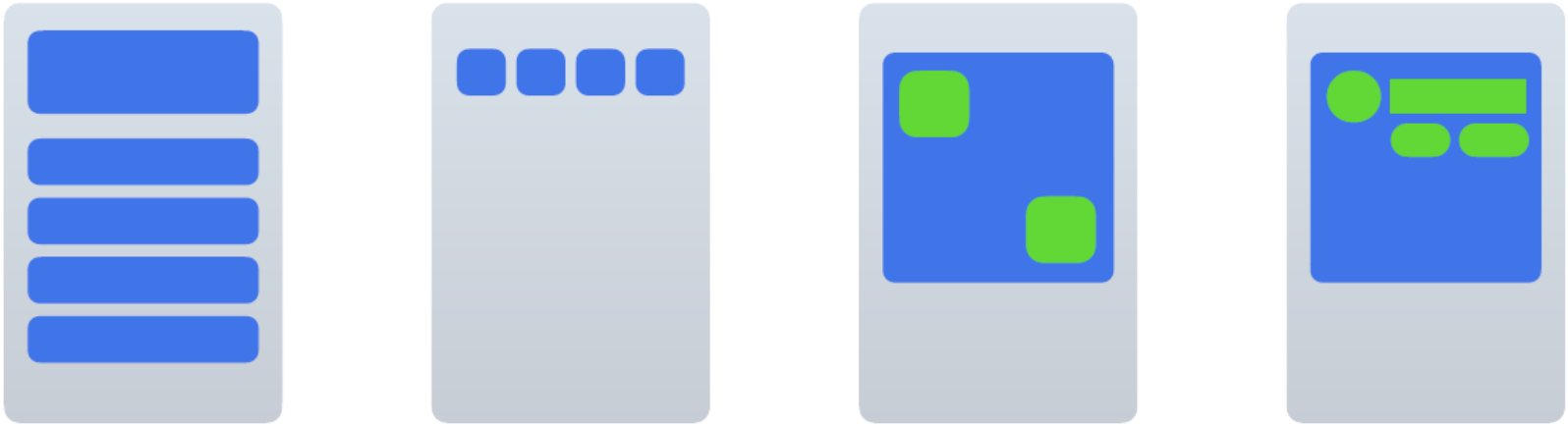
Layouts

Responsive UI

- Layout automatically **controls** the **size** and **placement (position** and **alignment)** of UI elements to create a **Responsive UI**
  - Flow provides an efficient way to **distribute space** among items while accommodating different screen sizes
  - Frees programmer from handling/hardcoding the sizing and positioning of UI elements
  - **Responsive UI =** When the screen is resized, the views reorganize themselves based on the rules of the layout

Layouts

- Use a Layout to **position** UI elements on the screen
- **`Row`** - position elements horizontally
- **`Column`** - position elements vertically
- **`Box`** - position elements in the corners of the screen or stack them on top of each other
- Use Constraint Layout (self-study) for complex layouts



Row & Column Example

- Group multiple basic layouts to create a more complex screen
- Use vertical or horizontal **Arrangement** to change the position of elements inside the Row or Column

```
@Composable
fun ArtistCard(artist: Artist) {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Image(/*...*/)
        Column {
            Text(artist.name)
            Text(artist.lastSeenOnline)
        }
    }
}
```

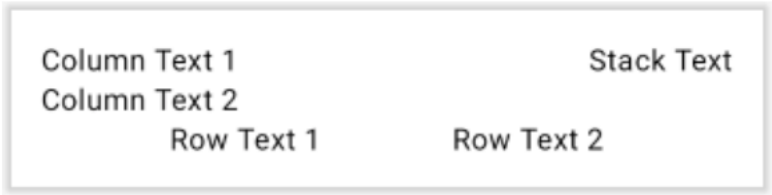
Alfred Sisley
3 minutes ago

Box Example

```
@Composable
fun ArtistAvatar(artist: Artist) {
    Box {
        Image(/*...*/)
        Icon(/*...*/)
    }
}
```

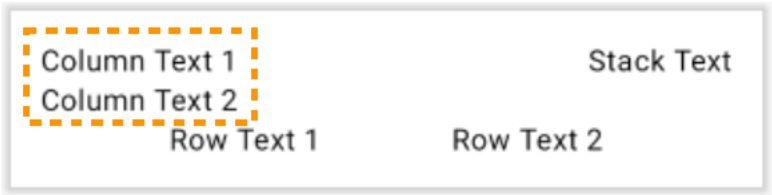
Box Example (1 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```



Box Example (2 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

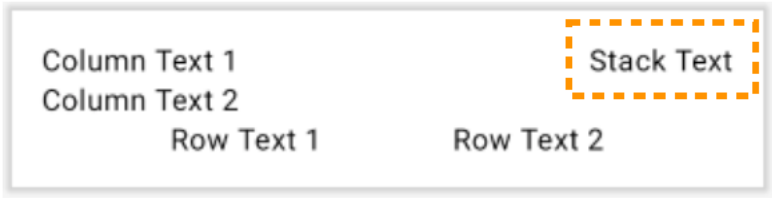


Box Example (3 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

Box Example (4 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```



Surface & Card

- A **Surface** can hold only one child with an option to add a border and elevation
  - Add a layout inside Surface to position multiple elements
- A **Card** is a just a Surface with default parameters

Responsive Layout

- Use the **`weight`** modifier in Row and Column layouts:
  - Use **weights** to change the **proportion** of the screen child elements will use
  - Distribute space among items in a container while accommodating different screen sizes
- `Modifier.`*`fillMaxWidth`*`()` fill available width
- `Modifier.`*`fillMaxHeight`*`()` fill available height
- `Modifier.`*`fillMaxSize`*`()` fill available width and height
- Use Constraint Layout (self-study) for more control for complex scenarios

Resources

- Jetpack compose tutorial

https://developer.android.com/jetpack/compose/tutorial

- Jetpack compose Code Labs

https://developer.android.com/courses/pathways/compose

- Jetpack Compose Playground - UI component examples

https://foso.github.io/Jetpack-Compose-Playground/

https://github.com/Foso/Jetpack-Compose-Playground

https://github.com/Gurupreet/ComposeCookBook

- Compose Samples

https://github.com/android/compose-samples